CSC 261/461
Database Systems

Eustrat Zhupa

November 20, 2018

Hashing Indexes

Hashes

- a *hash* function $h : K \rightarrow \{0, \ldots, B-1\}$ taking the key as argument, where B is the number of buckets.
- A *bucket* array indexed from 0 to $B-1$, holds the headers of B linked lists, one for each bucket of the array.
- If a record has search key K, store the record by linking it to the bucket list for the bucket numbered $h(K)$.

Hashing Indexes

Secondary-Storage Hashes

- A very large number of records must be kept *mainly* in secondary storage
- the bucket array consists of blocks, rather than pointers to the headers of lists.
- Records that hash to a certain bucket are put in the block for that bucket.
- If a bucket has too many records, a chain of overflow blocks can be added.
- a main-memory array of pointers to blocks indexed by the bucket number may be stored in main memory.

[1]

Hashing Indexes

Dynamic Hashing

- Most databases grow larger over time. If we are to use static hashing for such a database, we have three classes of options:
  1. Choose a hash function based on the current file size.
  2. Choose a hash function based on the anticipated size of the file at some point in the future.
  3. Periodically reorganize the hash structure in response to file growth.

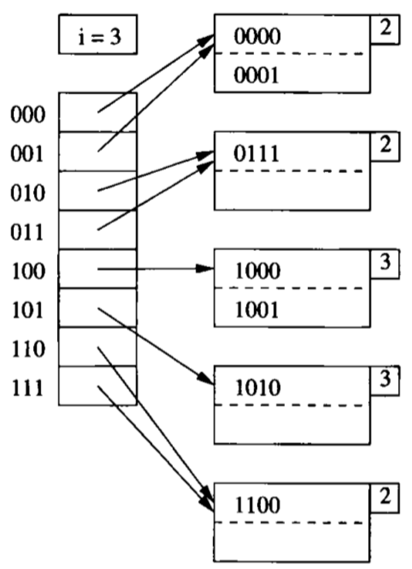
Hashing Indexes

Extensible Hash Tables

- There is a level of indirection for the buckets, array of pointers.
- The array of pointers can grow, but always a power of 2.
- certain buckets can share a block if the total number of records in those buckets can fit.
- The hash function computes for each key a sequence of k bits for some large k.
- the bucket numbers use some smaller number of bits, say i bits

Hashing Indexes

Extensible Hash Tables Example



i = 3
000
001
010
011
100
101
110
111
0000
0001
2
0111
2
1000
1001
3
1010
3
1100
2